VNB0227160

AN I[illegible]EMENTATION OF MLS ON A NE[illegible]ORK [illegible]WORKSTATIONS USING X.500/5[illegible]

James Davis, Doug Jacobson, Stephanie Bridges, and Ken Wright

Department of Electrical and Computer Engineering
Iowa State University
Ames, Iowa 50011
davis@iastate.edu

ABSTRACT

In this paper, we describe a project whose goal is to provide a secure distributed access control mechanism for user tasks in a heterogeneous network of computing resources. This is accomplished by implementing a UNIX-based Multi-Level Security (MLS) scheme where users and resources are labeled with a security level and a group. Access control is enforced by an access list server that uses X.500 directory and X.509 authentication services. Groundwork is laid for the next step of the project, which is to extend the security services for migrating tasks so that workstations are protected from security threats posed by incoming tasks, and also to protect tasks from threats originating from the workstation.

I. INTRODUCTION

In this paper, we describe a project that uses X.500 directory and X.509 authentication services to implement a MLS for a network of workstations. The example application described here concerns protecting information in an academic setting. System subjects (students, staff, administrators, etc.) and objects (tests, grades, course information, etc.) form a hierarchy in the classic MLS sense. We chose this application because it provides many examples of *need-to-know* sharing that are sometimes difficult to handle in hierarchical models of sharing. The next step of the project will shift the focus from access control at the user level to access control at the granularity of a task. We anticipate that this will allow us to construct a secure network of heterogeneous workstations operating as a loosely-coupled multiprocessor. Access control decisions will be made locally, but within the context of a network-wide security policy.

We have chosen to model a subset of information in an academic environment, focusing on the flow of information that takes place during the delivery of a course. As implementation of the project continues, the model can be extended in a hierarchical manner to include all courses in a department, then all departments in a college, etc.

In this example, the subjects are people involved in developing, delivering, or taking a course. The resources (or objects) include information about courses, such as the syllabus, homework assignments, tests, etc. Subjects and objects are organized into a MLS hierarchy by assigning a security label to every subject and object. By inspecting subject and object labels, we determine if an access request should be satisfied when the request is made. Some of the objects are more or less public (e.g., the course syllabus), and others are clearly private (e.g., grades). Interestingly, some objects play different roles through time. Consider the various stages of a test:

- during creation of the test, only the instructor can read or write the test
- when the test is distributed to students, only the student can write on it, and both the student and instructor can read it
- after the student submits the answers for evaluation, only the instructor can write on the test
- finally, after the test is returned to the student, disclosure of the results is at the student's discretion

The test example is particularly difficult to handle in many models because there is an implied change of ownership, and the test is declassified and then reclassified. (We handle need-to-know sharing by creating and deleting groups containing subjects who need to share information.)

A general diagram of the system is shown in Figure 1. Users generate requests for information by running programs on their local workstations. One node on the network handles the initial session authentication. Once authenticated, the user will receive a certificate that allows communication with the access list server that contains access permission information about subjects and objects. For example, to determine if student "Joe Smith" can read the "Midterm Exam" for course "CprE 384", the access list server will search the student list for Joe Smith, then the CprE 384 object list for the Midterm Exam, and compare the security levels. Fortunately, heuristics can be used to limit the search space for queries. We have modified standard UNIX libraries (e.g., libc.a) on the local workstation to generate queries of this type for library functions like *open*, *unlink*, etc. If the subject is not

This work was supported in part by National Security Agency grant MDA904-94-C-6111

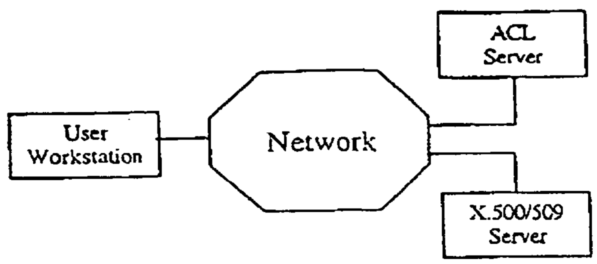


Figure 1. System Architecture

authorized for the requested access, the request will fail. This makes the MLS security model more or less transparent to user processes.

II. INFORMATION SHARING MODEL

In this section, we provide a few basic concepts on access control rules. Detailed information as it pertains to this project can be found in [1].

A. Access Control

Access control mechanisms control the flow of information so as to maintain the *principle of least privilege* [2]. That is, information should be disclosed only to the extent needed to carry out the specified task. It is common to divide schemes into two groups, namely, discretionary access control (DACs) and mandatory access control (MACs). In practice, a hybrid scheme that has some degree of discretionary control within the context of mandatory control is very useful.

Discretionary access control allows users to make decisions on how information is shared. For example, in the UNIX environment, the owner of a file has the ability to keep it private, or share it with others in his group, or make the file accessible by everyone on the system. DAC schemes provide the greatest flexibility for users, but in the absence of enforced system-wide rules on sharing, it is not possible to guarantee non-disclosure of sensitive information.

Mandatory access control specifies a set of rules that are enforced system-wide. These are usually implemented by the operating system (or hardware) so that users cannot violate a sharing rule. A simple example of a mandatory policy would be a model where subjects and objects are categorized as being Top Secret, Secret, Confidential, and Unclassified. A person with a Secret clearance would be able to read documents that are classified as Secret, Confidential, or Unclassified, but the person cannot read Top Secret documents. If sharing rules are correctly implemented, we can guarantee that sensitive information is not disclosed.

Note that in the above example, a person with a Secret clearance has access to all documents that are Secret. In practice, we would like to better control disclosure of information so that it is disclosed on a *need-to-know* basis. One common extension to a hierarchical model is the addition of *groups* or *compartments*. The natural analogy here is that of a multi-person project: team members with different talents (and security clearances) are brought together to accomplish a task, and information pertaining to that task is known only to members of the group. That is, someone with a Secret clearance cannot automatically obtain Secret information about the group's project. Need-to-know sharing between subjects who are not in the same group can be facilitated by simply creating a new group with those subjects as its only members. Because disclosure is explicitly documented by the model (and transitivity is not allowed), unwanted disclosure is detected when groups are established or new members are added. (We assume conflict of interest is handled administratively at that time also.)

B. The BLP Model with Groups

At the core of many information flow models is the work on mandatory access control rules by Bell and LaPadula [9], referred to as the BLP model. The BLP model prevents *write-down* and *read-up* in a multilevel security system with two simple MAC rules that determine if the access is appropriate by comparing the subject's and objects's security labels.

While the BLP MAC rules address the problem of sharing information between security classes, they do not limit information flow between subjects in the same security class. For example, in a project course with groups of students, each group may choose to protect private information from another group, even though all subjects are students and have the same clearance. The BLP rules can be modified to handle this restriction on sharing, as follows.

The *Simple Security Property* becomes:

A subject S can read object O if and only if:
$SL(S) \geq SL(O)$ and $(group(O) \cap group(S) \neq \varnothing)$

That is, subject S can read object O only if the security level of S dominates the security level of O, and they are members of a common group. It's implied that subjects and objects can simultaneously belong to multiple groups.

The **-Property* becomes:

A subject S can write object O if and only if:
$SL(S) \leq SL(O)$ and $(group(O) \cap group(S) \neq \varnothing)$

C. Adding Discretionary Access[illegible]

We rewrite the access rules o[illegible]re time to add the concept of discretionary access so that an owner of an object can selectively disclose the object to those subjects who meet the requirements of the mandatory access portion of the access rules. The Simple Security Property then becomes:

A subject S can read object O if and only if:

1. subject S owns object O
   or
2. SL(S) ≥ SL(O) and (group(O) ∩ group(S) ≠ ∅) and discretionary access allows reading for all groups in group(O) ∩ group(S)

The *-property can be rewritten as:

A subject S can write object O if and only if

1. subject S owns object O
   or
2. SL(S) ≤ SL(O) and (group(O) ∩ group(S) ≠ ∅) and discretionary access allows reading for all groups in group(O) ∩ group(S)

In our application, the major use of groups is to separate students in different courses, i.e., there is one or more groups for each course.

III. USING X500 DIRECTORY SERVICES AND X509 AUTHENTICATION

This section provides a brief background on the X.500 directory and X.509 authentication services used in this project.

A. X.500 Directory Overview

X.500 [3] is the OSI (Open Systems Interconnection) standard which defines a distributed directory service. The X.500 series of recommendations describes a collection of open systems which cooperate to provide information to users about a set of objects. The information held in the directory, called the Directory Information Base (DIB), can be used by many different applications for different purposes, and accessed by a wide range of clients. Different applications may require different kinds of information, so a large number of object classes and attribute types have been defined. The directory works to make sure that all entries adhere to the rules to prevent entries from having the wrong types of attributes for its object class, attribute values being of the wrong type, and entries from having the wrong type of subordinate entries.

Directory Model

The directory consists of a set of Directory System Agents (DSAs), with a Directory Information Base (DIB) which is distributed among the DSAs. A set of Directory User Agents (DUAs) access the DSAs to retrieve data from the DIB. The DSAs communicate with each other with the Directory System P[illegible]l (DSP); a DUA communicates with a DSA via the Directory Access Protocol (DAP).

A user, represented by a DUA, can query a DSA for information. The DSA may either return the requested information, chain the query to another DSA, return a referral to another DSA to the DUA, or return an error. This operation is shown in Figure 2. Note that DSA C may return the referral from DSA B directly to the DUA if it chooses. In this case, a connection from the DUA to DSA B would be made. DSA B might also have chosen to chain the request on to DSA A and would have then returned the information directly to DSA C. In this case the Request/Result transfer between DSA C and DSA B would have been replaced by a chained request from DSA A to DSA B. These operations all occur without any interaction from the user represented by the DUA, who will only receive the result from the DUA.

Information Model

The Directory is structured as a hierarchical tree of information where entries appear at all nodes except the root node. Each entry is a collection of information about one object; there may be only one entry corresponding to each unique object.

The entries are divided among servers in a geographical and organizational distribution. Entries are named according to their position in the hierarchy by a distinguished name (DN). Each component of the DN is called a relative distinguished name (RDN). An object may have one or more alias entries, so that the structure of the tree is not strictly hierarchical. In the DIT structure in Figure 3 each of the rectangular boxes represents a directory entry. By concatenating the RDN of each entry as we traverse the tree, we uniquely identify the entry with its distinguished name. By traversing the tree along the path shown in Figure 3, we would have the distinguished name {C=US, O=Iowa State University, OU=Electrical & Computer Engineering, CN=Jane Baker}.

Each entry is a collection of attributes, which describe the object which the entry references. Each attribute consists of an *attribute type*, which defines the class of information given by an attribute, and one or more *attribute values* which are the particular instances of that class in the directory. At most one of the values of an attribute may be designated as a *distinguished value*, which means it will appear in the RDN of the entry. More details on X.500 as it was set up for this project can be found in [4].

B. X.509 Authentication Overview

A framework for providing authentication services is described in the X.509 [5] standard. This standard provides a framework of authentication services by the directory to its users. These users can include the directory itself as well as other applications and services. The directory can be used as a place where authentication information can be stored and easily retrieved.

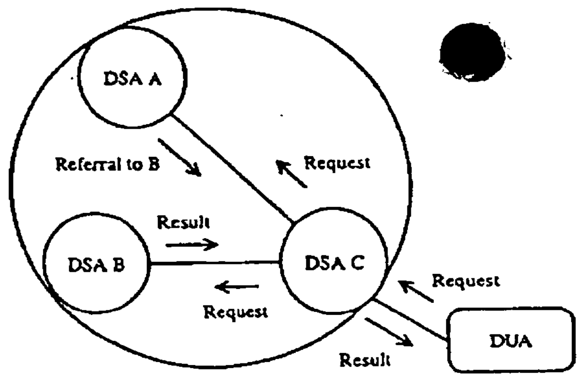


Figure 2. DSA Functional Model

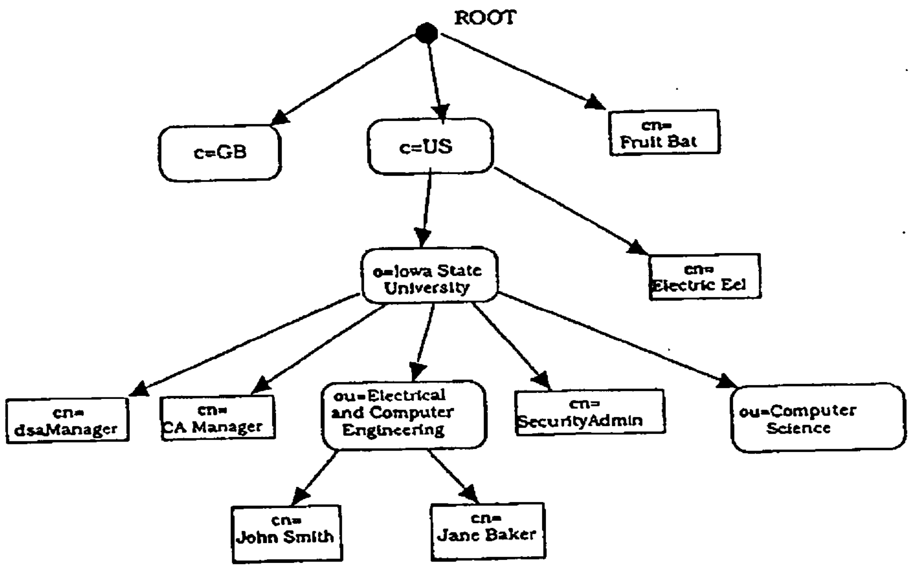


Figure 3. Example ISU DIT

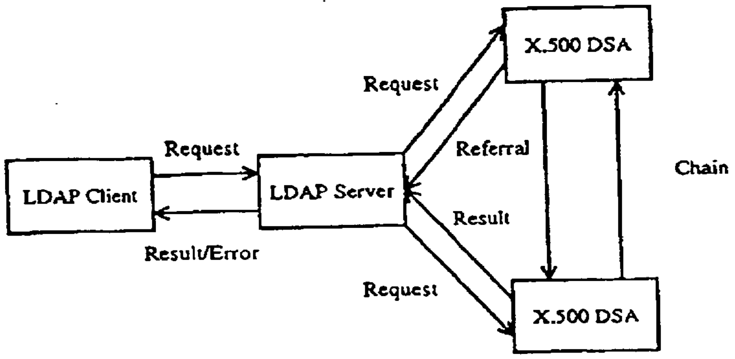


Figure 4. Relationship between LDAP and X.500

The recommendation specifie[illegible] form of the authentication information held by t[illegible]ectory, describes how authentication information may be obtained from the directory, states the assumptions made about how the information is formed and placed in the directory, and defines three ways in which applications may use this information to support authentication and describes ways other security services may be supported by authentication.

We use strong authentication, which involves the use of public-key cryptography to perform authentication. Each user has a pair of keys, one public and stored in the directory, and one secret. Each user is identified by his possession of his secret key. For user A to verify that a communication partner B is in possession of user B's secret key, A must be in possession of user B's public key. User A must be able to obtain B's public key from a source that A and B both trust.

Recommendation X.509 describes a procedure where a certification authority (CA), trusted by all the users it serves, certifies the public keys by signing them to produce a certificate. The certificates have the property that anyone with access to the CA's public key can recover the public key of the user, and that no one other than the CA can alter the contents of a certificate without it being detected. The public key of a CA is known to all users. This provides some measure of assurance that the public key of a CA is, indeed, his public key and can therefore be trusted. Since the certificates are unforgeable, they can be placed in the directory without any further protection.

In a large community with many users, it may not be practical for all users to have certificates managed by the same CA. The standard provides for a way for CAs to securely exchange their own public keys so that a user can build a certification path to any other user as long all the CAs in the chain have created certificates for each other. All the certificates of CAs by CAs must appear in the directory and the user must know how they are linked to form a path to make use of them. The standard has specified a hierarchical relationship among the CAs, so that navigation is straightforward.

X.509 specifies three alternate authentication procedures which make use of public key certificates, namely one-way, two-way, and three-way authentication. It is assumed in all three procedures that the two users know the other's public key, either because they have obtained it from the directory or because the certificate was included in the initial message from each side. We use the two-way protocol which allows both parties to mutually authenticate each other.

C. Lightweight Directory Access Protocol

The Lightweight Directory Access Protocol (LDAP) [6,7] was developed to provide access to the X.500 directory without incurring the high overhead costs associated with DAP. The protocol is targeted for simple management and bro[illegible]pplications to provide simple read/write access to the X.500 Directory. The protocol elements are carried directly over TCP or some other transport, thus bypassing much of the session/presentation overhead. The LDAP protocol provides a simplified interface to an X.500 Directory, omitting some little used features and emulating some operations with others.

LDAP functions as a client-server with an LDAP server accepting requests from LDAP clients and translating those requests into X.500 requests to an X.500 DSA. As shown in Figure 4, an LDAP client sends a query to the LDAP server who then formats and sends it on to the X.500 DSA. The DSA may either return a result itself, chain the request to another DSA, or return a referral. The LDAP server will never return anything to the client except a result or error.

IV. IMPLEMENTATION OF THE ACADEMIC MODEL

The application focuses on information sharing in an academic environment. Components of the model include instructors, teaching assistants, students, courses, and course materials (tests, homework, solutions, grades, etc). The model is hierarchical and can be scaled to a large organization. This example presented here will focus on information sharing for one course.

There are four components in the model: security level, groups, people (subjects), and objects. All users and all objects are labeled. A security label is defined as a tuple consisting of the security level and the group list for that object or subject. The components are defined as follows:

- Five security levels have been defined, in order of increasing need for security: Unclassified, student, Grader/TA, Instructor, Course Supervisor.
- Each person will have a name which is unique throughout the system and may belong to more than one group. Additionally, subjects may have different security levels in different groups.
- A group can be comprised of any collection of users; each group has one owner who is responsible for maintenance.
- An object refers to a collection of information, for example "Exam#1". Each object is owned by one person, and is accessible to a specified list of groups. An object has one security level which cannot be higher than that of the owner's level in any group. Objects do not necessarily inherit membership in the owner's groups.

A hypothetical course description is shown in Figure 5, with associated groups, subjects, and objects. In the next section, we will describe how the X.500 Directory is used to store information about the subjects in the model and how it is integrated with the security access server to allow access to the objects in the model.

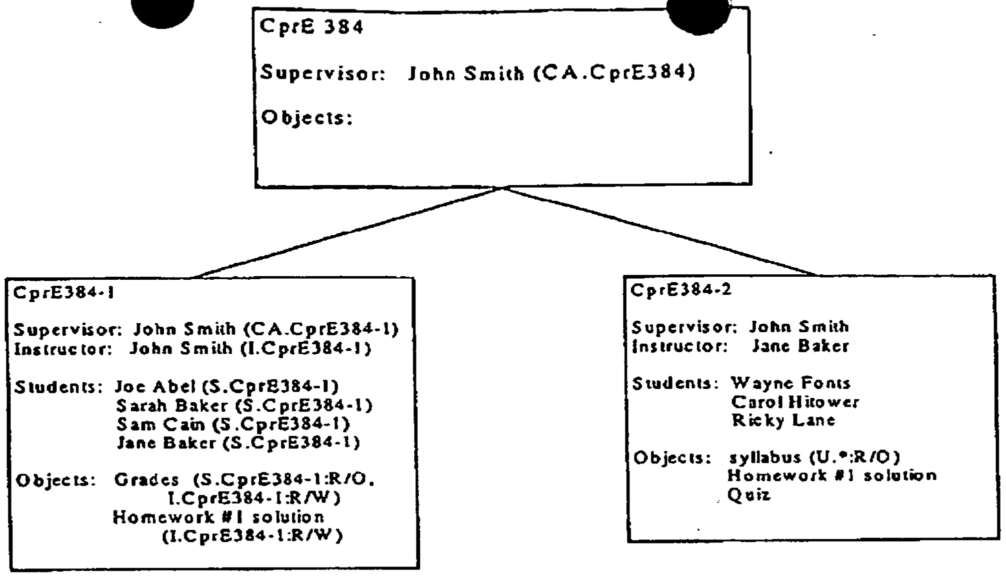
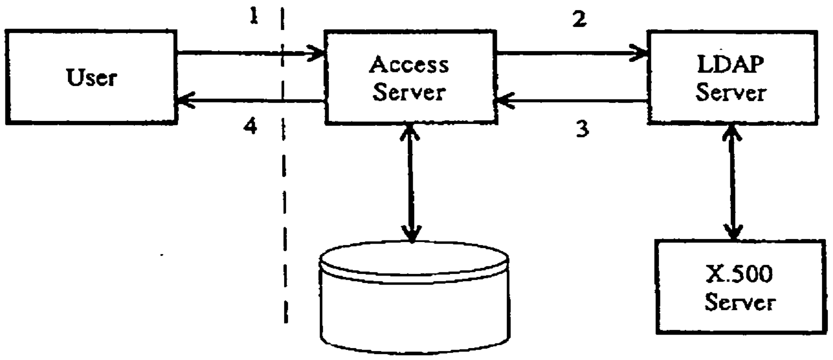

CprE 384
Supervisor: John Smith (CA.CprE384)
Objects:
CprE384-1
Supervisor: John Smith (CA.CprE384-1)
Instructor: John Smith (I.CprE384-1)
Students: Joe Abel (S.CprE384-1)
Sarah Baker (S.CprE384-1)
Sam Cain (S.CprE384-1)
Jane Baker (S.CprE384-1)
Objects: Grades (S.CprE384-1:R/O, I.CprE384-1:R/W)
Homework #1 solution (I.CprE384-1:R/W)
CprE384-2
Supervisor: John Smith
Instructor: Jane Baker
Students: Wayne Fonts
Carol Hitower
Ricky Lane
Objects: syllabus (U.*:R/O)
Homework #1 solution
Quiz

Figure 5. Example course description



1. UserRequest={{"Jane Baker","CprE384-1 Grades",READ,}SIGNATURE}
2. AccessServerRequest= {SEARCH_DSA,"Jane Baker"}
3. LDAPSreverReply= {cn#userClass#userCertificate}
4. AccessServerReply= {"CprE384-1 Grades" | "Access denied"}

Figure 6. Sample Access Query

A. Directory Model

Entries for each of the subjects associated with the example have been entered into the "Electric Eel" DSA. The attribute values as they are stored in the EDB for "cn=Jane Baker" are shown in Table 1. The groups that a person belongs to are stored as a space-delimited list in the *userClass* attribute. Note that only the Security Administrator is given permission to alter this attribute since users should not have the ability to change their group membership. Each of the subject entries contains a *userCertificate* attribute which contains a signed copy of their public key.

The *userCertificate* is stored in the EDB according to the BNF description of the attribute syntax shown in. These are issued by the CA for the department and users are given an encrypted copy of their private key which may be used for signing requests to the access server.

able 1. Directory entry for Jane Baker

Attribute	Value
objectClass	strongAuthenticationUser & newPilotPerson & quipuObject
commonName	Jane Baker
surname	Baker
userPassword	{CRYPT}PF\40QFW
userCertificate	md2WithRSAEncryption#{ASN}050000# 2784bbc9b1a3223cf01b6f615d7b6a5204ecff93e21aef304d97668cfc01d0fb733625337ba6c 092778774b82c80e057e443c5ec55543dc4fcd29791a8d78b95# c=US@o=Iowa State University@ou=Electrical and Computer Engineering# c=US@o=Iowa State University@ou=Electrical and Computer Engineering@cn=Jane Baker# md2WithRSAEncryption#{ASN}050000# 0#{ASN}021100b0669fb584d45ad64735acd8569e77cf00# 960603165052Z#960703165052Z#rsa#{ASN}0202020000# 304702405977023c32493e2c8d6b5770861c0f261007df8537d70215c6782d93f7ac2d08e67 57c28d7e9a6313e0a073ab0980807079fa001f6d28b74e4c59815a34a16eb0203010001#
userClass	CprE384-2(Instructor) CprE384-1(Student)
lastModifiedTime	Mon Jun 3 11:50:56 1996

lastModifiedBy	countryName=US
	organizationName=Iowa State University
	commonName=CA Manager

accessControlList	others can read the child self can write the child group (c=US@o=Iowa State University@cn=dsaManager) can write the child others can read the entry self can write the entry group (c=US@o=Iowa State University@cn=CA Manager) can write the entry group (c=US@o=Iowa State University@cn=dsaManager) can write the entry others can read the default self can write the default others can compare the attributes: userClass others can read the attributes: userClass group (c=US@o=Iowa State University@cn=SecurityAdmin) can write the attributes: userClass others can read the attributes: userCertificate,objectClass self can write the attributes: userCertificate,objectClass group (c=US@o=Iowa State University@cn=CA Manager) can write the attributes: userCertificate,objectClass

Certificates are issued no longer than the duration of a semester. If a person becomes a member of none of the groups managed by this application (for example, by dropping out of a class), his certificate may be revoked in addition to his being removed from the group. Certificates would be reissued at the beginning of every semester when groups are reassigned. Certificates may also be revoked at any time for other administrative reasons, for instance in the case when a user's private key is compromised.

B. Access Server Model

When a person makes a request to access an object, several things must happen before access is granted. Figure 6 depicts the protocol described in the following example.

Suppose user "Jane Baker" wishes to access the object "Grades" in group "CprE384-1". "Jane Baker" creates a request signed with her private key containing her name, and the name of the object she wants to access, in this case "CprE384-1 Grades".

Upon receiving the request from "Jane Baker", the access server sends a request to the LDAP server to search the X.500 DSA for the user named in the request. The LDAP server retrieves a set of attributes from the X.500 DSA and returns a data structure containing the attribute values to the access server. Details on our port of LDAP for this project can be found in [4].

The access server retrieves the public key from the user's Certificate which it received in the LDAP response and uses it to verify the identity of the user sending the request. If the identity is verified, the access server compares the returned userClass string to the groups allowed access to the requested object. If the requested access is granted, the server returns the object to the user, or if access has been denied, an appropriate error message.

Note that these transactions are transparent to the user. The initial request for access to an object was generated by an I/O request (e.g., opening a file) made by an application program the user ran on his/her local workstation.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we briefly described how a MLS policy can be implemented on a network of workstations. Directory services were provided by X.500 and authentication services by X.509. We ported LDAP software to help reduce the overhead in using X.500 this way. The access list for subjects and objects is stored on a workstation, and is maintained and queried by small c programs running on the workstation. As a proof of concept, end-users can become part of the network MLS by linking their programs to a special library that replaces systems services with equivalent services that also invoke the MLS software.

To date, the major time commitment has been making the ISODE software operational. Once running, though, it has required minimal maintenance.

There are several experiments and enhancements that we are currently pursuing:

- Continue testing the distributed X.509 software.
- Related, migrate to IC RC3 [8], which provides the functionality of the 1993 standards including replication, access control, and authentication.
- Continue to fill the X.500 database, and scale the problem size up to the point where available network bandwidth becomes a problem
- Investigate distributing the access list to improve performance
- Patch a UNIX kernel instead of modifying library routines. This would force all users to run under the MLS

VI. BIBLIOGRAPHY

1. Wright, Kenneth, "Multilevel security access control in an open systems environment", MS thesis, Iowa State University, July 1996.
2. Pfleeger, Charles, *Security in Computing*, Prentice-Hall, 1989.
3. Recommendation X.500 ISO/IEC 9594-1, Information Technology, Open Systems Interconnection, The Directory: Overview of Concepts, Models, and Services, 1993.
4. Bridges, Stephanie, "An implementation of multi-level security for distributed applications using X.500/X.509", MS thesis, Iowa State University, July 1996.
5. Recommendation X.509 ISO/IEC 9594-8, Information Technology, Open Systems Interconnection, The Directory: Authentication Framework, 1993.
6. Yeong, W., T. Howes, and S. Kille, "Lightweight Directory Access Protocol", RFC 1777, Performance Systems International, University of Michigan, ISODE Consortium, March 1995.
7. Young, A., "Connection-less Lightweight X.500 Directory Access Protocol", RFC 1798, ISODE Consortium, June 1995.
8. *Introduction to the ISODE Consortium Products and Services, release 3.0*, ISODE Consortium, April 1995.
9. Bell, D. and L. LaPadula, "Secure Computer Systems: Mathematical Foundations and Model", *Mitre* Report. MTR 2547 v2, Nov 1973.